DW_lp_pipe_mgr

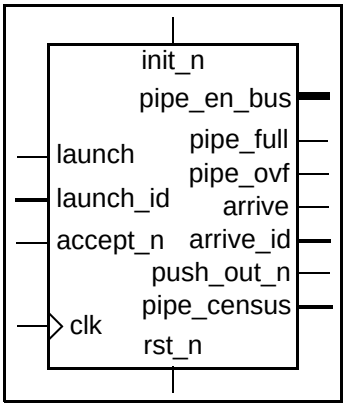
Low Power Pipeline Manager

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History

- Saves power by only enabling stages as needed
- Parameter controlled pipeline stages
- Flow control to interface directly to FIFO
- Flow control interfaces to another managed pipe
- Bubble removal extends depth of subsequent FIFO by pipe depth
- Parameter sized identifier tracks data operations



Description

The DW_lp_pipe_mgr manages the enabling of register stages of a pipeline based on launch requests that track the progress of a pipelined operation. Enabling stages of the pipeline only when they need to be clocked reduces dynamic power and is further enhanced through clock gate insertion (which requires the Power Compiler product). Launch requests can be accompanied by a launch ID which can be used as a tag to identify operation results as the pass out of the pipe. A census output monitors the number of operations in the pipe.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock source
rst_n	1 bit	Input	Asynchronous reset (active low)
init_n	1 bit	Input	Synchronous reset (active low)
launch	1 bit	Input	Active high control input to launch data into pipe
launch_id	*id_width* bits	Input	ID tag for data being launched (optional)
pipe_full	1 bit	Output	Status flag indicating no available slot in pipe
pipe_ovf	1 bit	Output	Error flag indicating pipe overflow (data lost)
pipe_en_bus	stages	Output	Bus of register enables (one per pipe stage)
accept_n	1 bit	Input	Active low flow control input for data out of the pipe (interface to output FIFO)
arrive	1 bit	Output	New data available status output
arrive_id	*id_width* bits	Output	ID tag for data arriving at pipe output (optional)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
push_out_n	1 bit	Output	Active low output used to interface with output FIFO (optional)
pipe_census	ceil(log2(*stages* + 1)	Output	Output bus indicating the number of operations currently in the pipe

Table 1-2 Parameter Description

Parameter	Values	Description
stages	1 to 1023 Default: 2	Number of logic stages in the pipeline
id_width	1 to 1024 Default: 2	Width of `launch_id` and `arrive_id`

Table 1-3 Synthesis Implementations

Implementation Name	Implementation	License Feature Required
rtl	Synthesis model	■ DesignWare (P-2019.03 and later) ■ DesignWare-LP[a] (before P-2019.03)

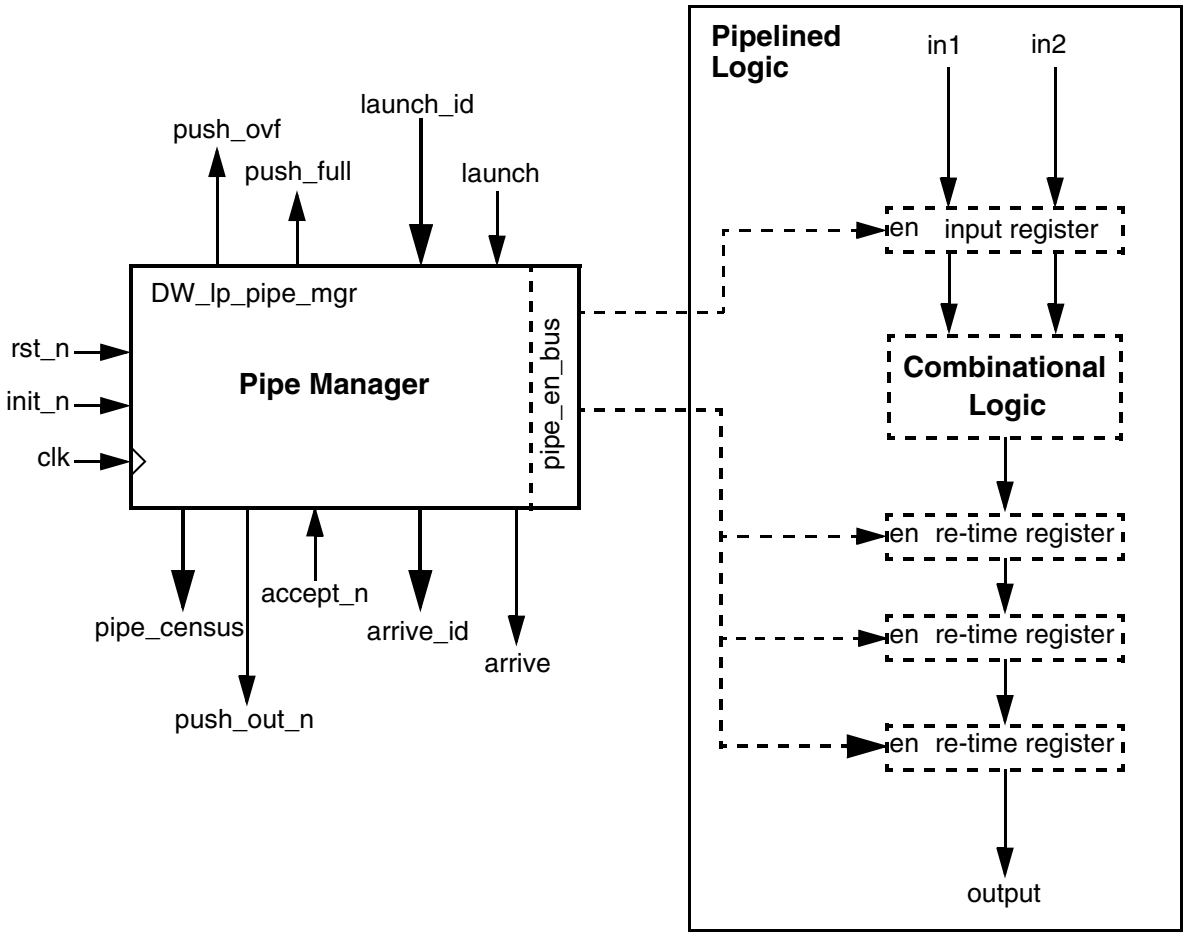
a. For Design Compiler versions before P-2019.03, see "Enabling minPower" on page 9.

Table 1-4 Simulation Models

Model	Function
DW03.DW_LP_PIPE_MGR_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_lp_pipe_mgr_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_lp_pipe_mgr.v	Verilog simulation model source code

Block Diagram

Figure 1-1 DW_lp_pipe_mgr Basic Block Diagram



Functional Description

Pipelining

The DW_lp_pipe_mgr is configurable embedded pipeline register levels. Setting the value for the parameter `stages` determines the number of pipeline register level(s) that are controlled. Therefore, depending on the parameter number of stages, the number of clock cycles for “in1” and “in2” to propagate to “output” varies.

This DW_lp_pipe_mgr is designed to make it easy to pipeline your logic using the register retiming features of Design Compiler (DC). It also contains parameter controlled input and output registers which will stay in place at their respective block boundary - they are not allowed to be moved by DC register retiming features. The input and output registers are not available when using DC versions earlier than A-2007.12.

The parameter `stages` refers to the number of logic stages desired after register retiming is performed. The number of register levels is not necessarily the same as the number of logic stages. If either an input register or output register is specified, the number of register levels is the same as the number of logic stages.

Pipeline Control and Power Savings

The DW_lp_pipe_mgr pipeline manager provides pipeline control logic (see Figure 1-1 on page 3) that monitors the activity. In cases where there is inactivity on a particular register level of the pipeline, the pipeline control disables those levels to promote power savings. Furthermore, if using the Synopsys Power Compiler tool, the presence of the pipeline control and its wiring to the pipeline register levels provides an opportunity for increased power reduction in the form of clock gating.

Along with the potential power savings that the pipeline control provides, it can be utilized to improve performance in cases where intermittent launch operations are present and there contains first-in first-out (FIFO) structures upstream and downstream of the pipelined logic. The handshake is made between the DW_lp_pipe_mgr and the external FIFOs via the accept_n and pipe_full ports. Effectively, the DW_lp_pipe_mgr can be considered part of the external FIFO structures. The performance gain comes when inactive (bubbles) stages are detected. These pipeline 'bubbles' are removed to produce a contiguous set of active pipeline stages. The result is empty pipeline slots at the head of (or entering) the DW_lp_piped_mult pipeline for new operations to be launched. Advancing the shifting of operations through the pipeline when a valid “output” is available (`arrive` = 1) is controlled by the accept_n input. When the pipeline is full of active entries, the `pipe_full` signal is 1. To disable this feature in cases where no external FIFOs are present, set the `accept_n` input to 0 which will effectively eliminate any flow control. At the same time, the `pipe_full` output would always be 0.

To assist in tracking of “launched” operands, the pipeline control logic provides interface ports called `launch_id` and `arrive_id`. The launch_id input is assigned a value during an active launch operation. Given that launch_id values are unique in successive launch operations, the “output” can be distinguished from one another with the assertion of arrive and the associated arrive_id. The arrive_id is the launch_id from the originating launch that produced the valid “output.”

No Pipeline Register Levels Specified

In cases where no pipelining is required through the pipelined logic (*stages* = 1), the pipeline control flow control handshaking/status signals still remain active and meaningful with one exception. The `pipe_census` output, which is intended to count the number of active pipeline register levels, becomes irrelevant and is fixed to 0.

Timing Waveforms

Figure 1-2 shows the DW_lp_pipe_mgr accepting (`accept_n` = 0) until empty (*stages* = 7).

Figure 1-2 Accept until empty

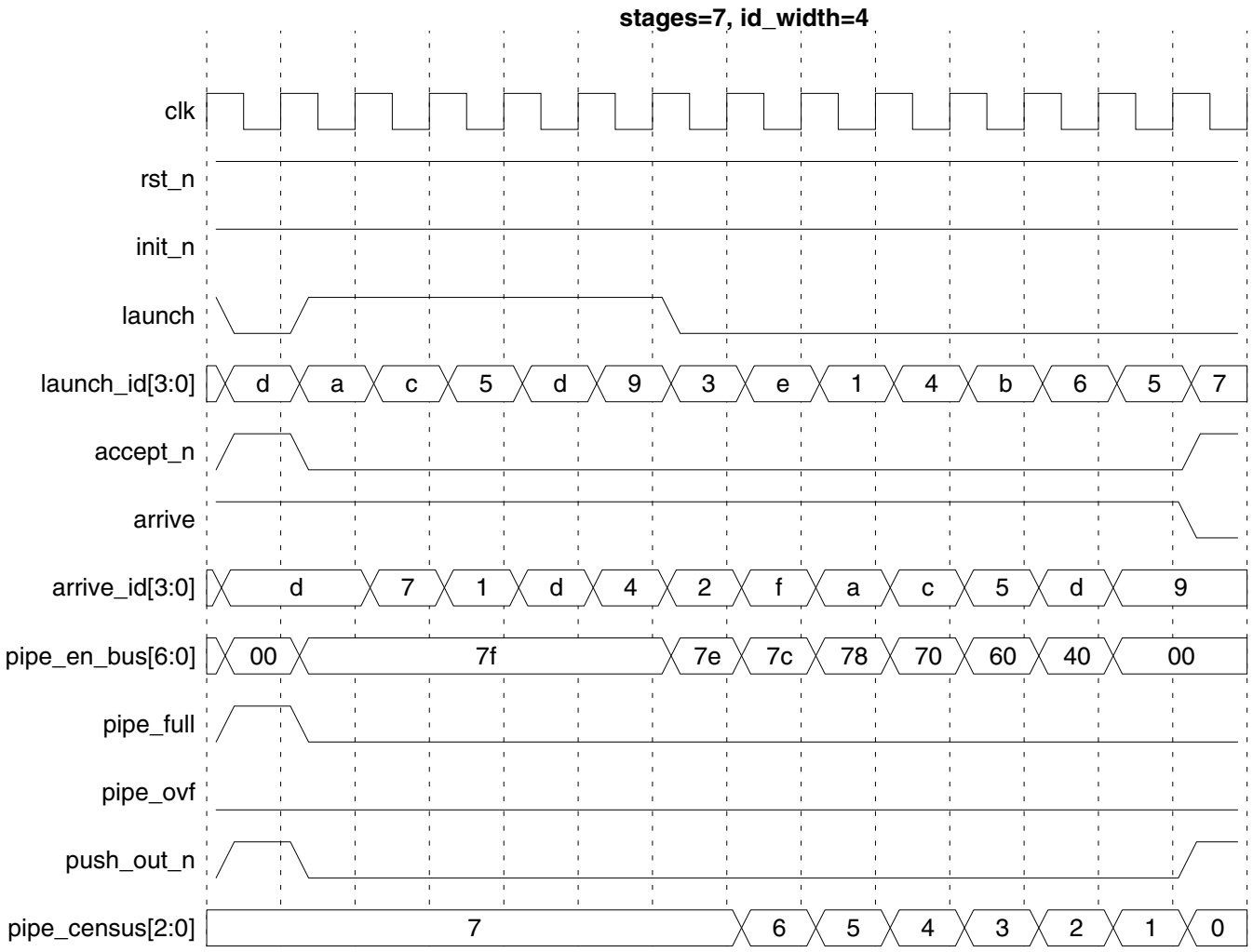


Figure 1-3 shows `launch` and `accept_n` changing simultaneously (*stages* = 7).

Figure 1-3 launch and accept_n Change Simultaneously

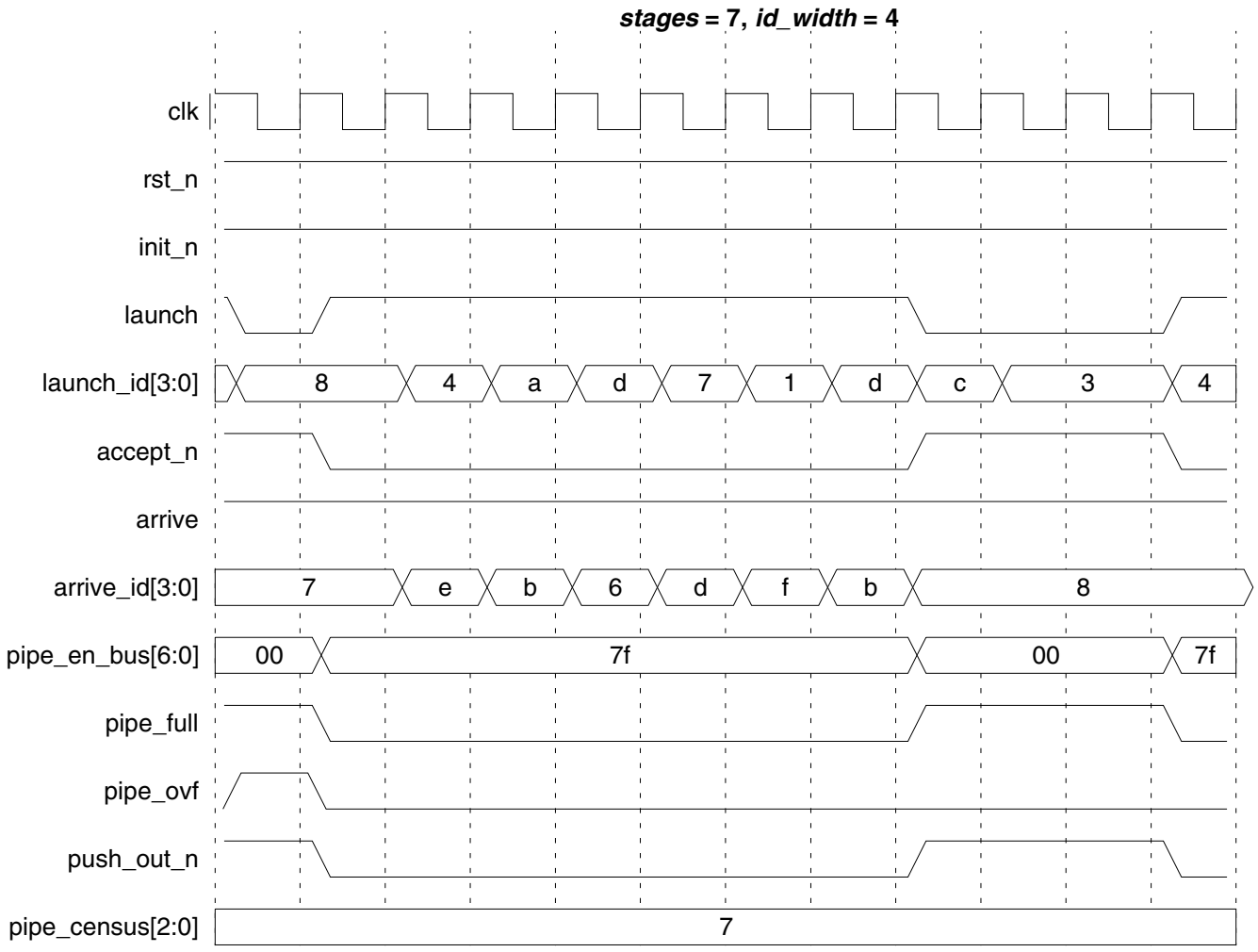


Figure 1-4 shows `launch` to `pipe_full` and then overflow (*stages* = 7).

Figure 1-4 Launch to Full, then Overflow

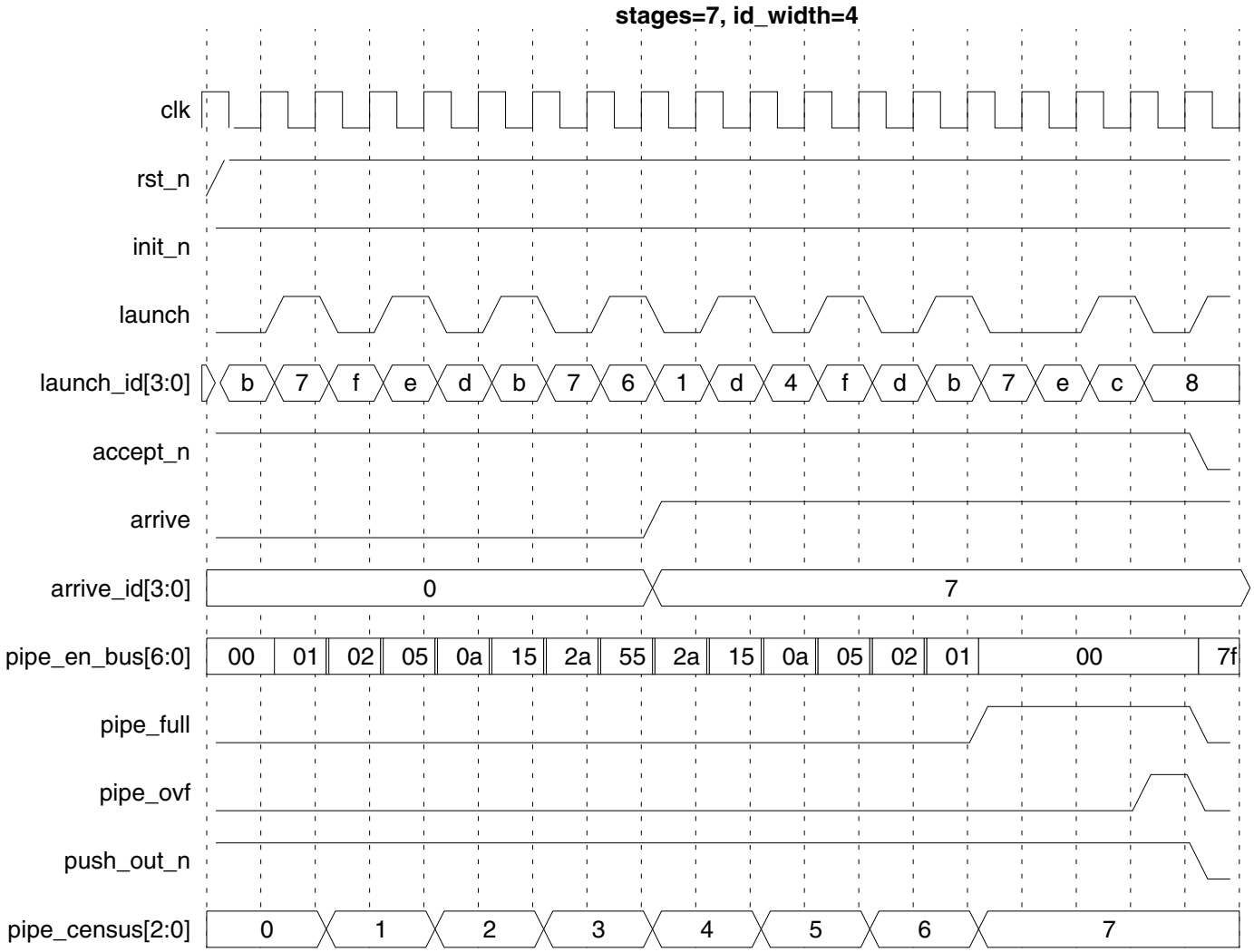


Figure 1-5 shows asynchronous reset (`rst_n`) behavior of DW_lp_pipe_mgr (*stages* = 7).

Figure 1-5 Asynchronous reset behavior (rst_n asserted)

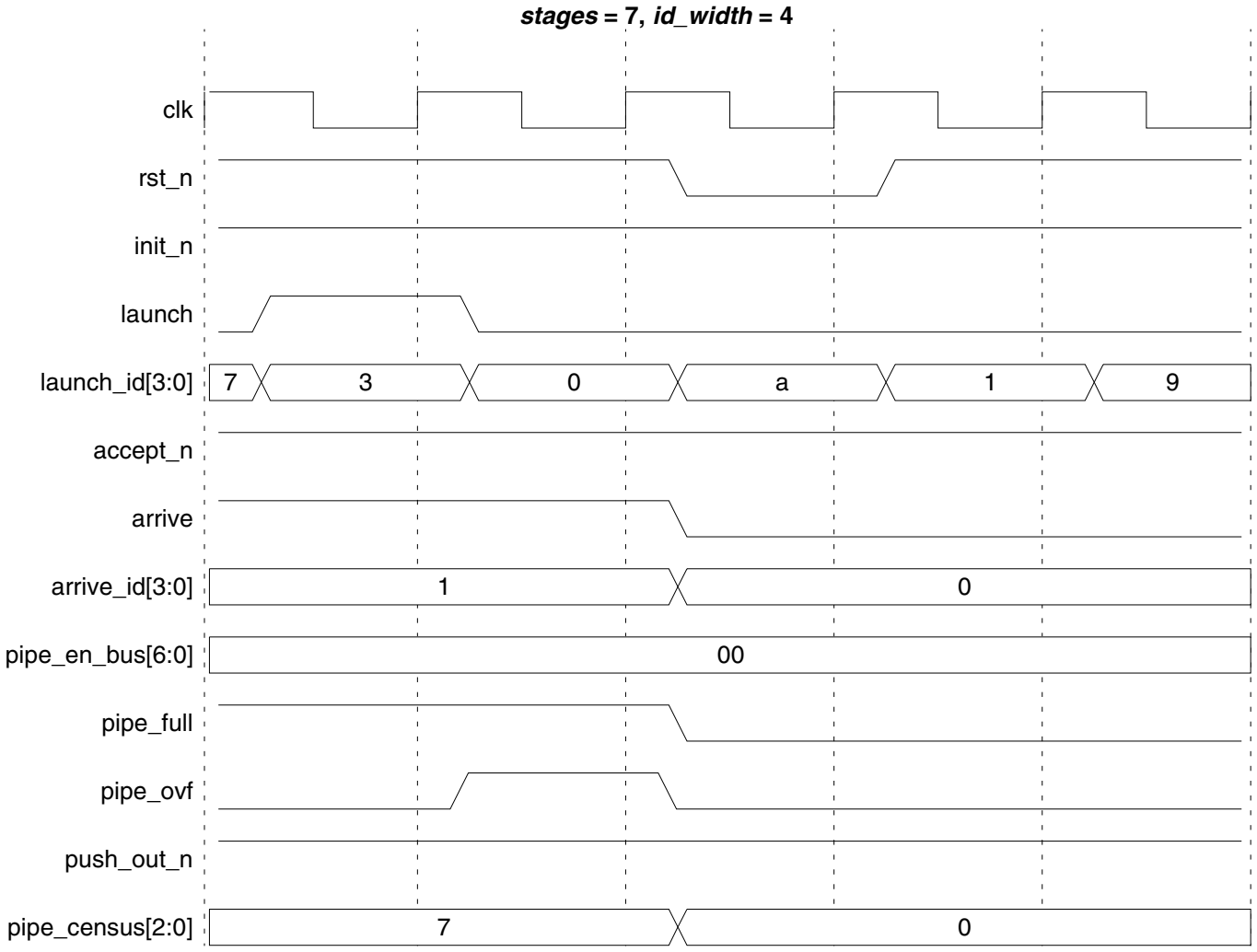
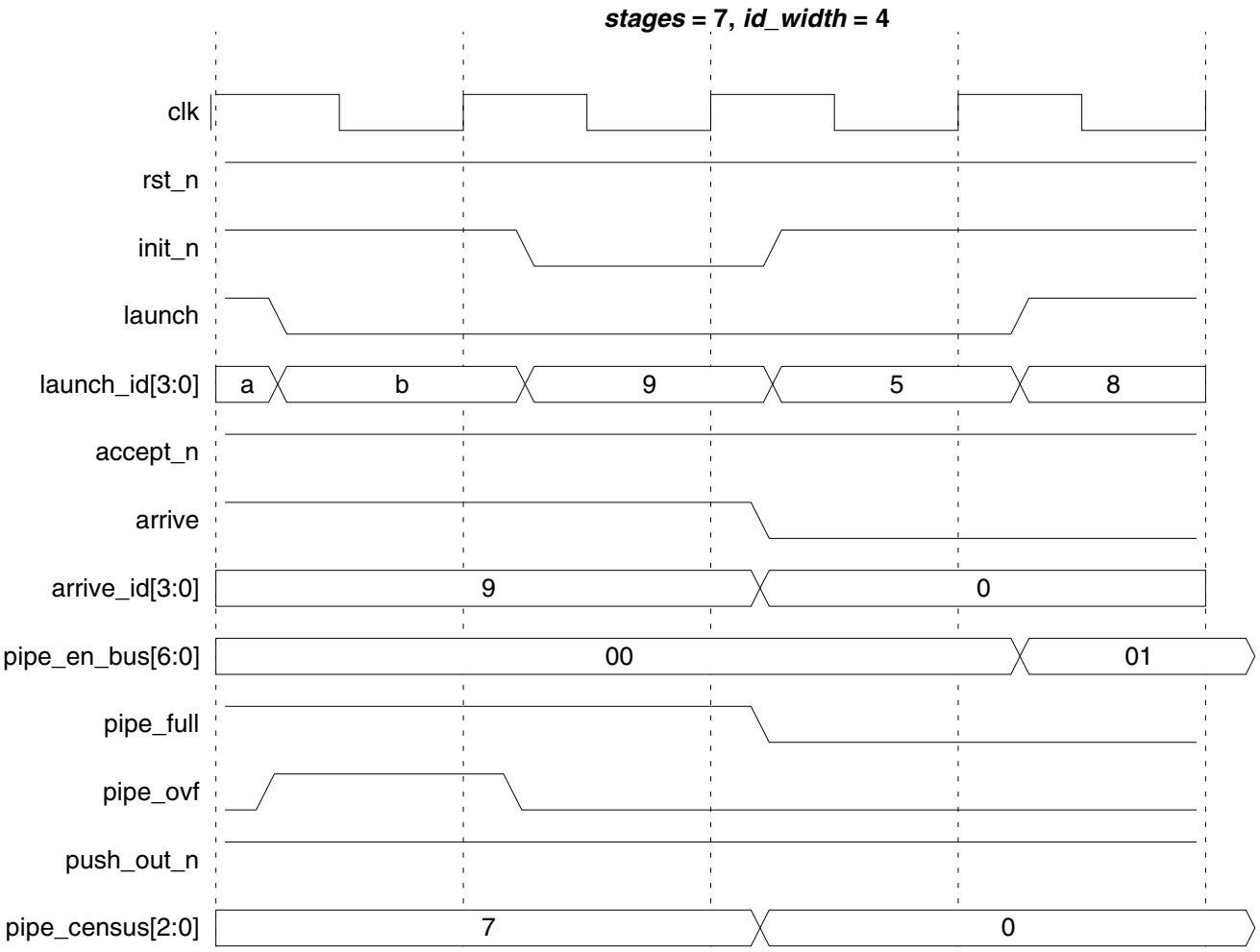


Figure 1-6 show synchronous reset (`init_n`) behavior of DW_lp_pipe_mgr (*stages* = 7).

Figure 1-6 Synchronous Reset Behavior (`init_n` Asserted)



Enabling minPower

In Design Compiler (version P-2019.03 and later) and Fusion Compiler, you can instantiate this component and use all its features without special settings.

For versions of Design Compiler before P-2019.03, enable minPower as follows:

```
set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
set link_library {* $target_library $synthetic_library}
```

Related Topics

- DesignWare Building Blocks User Guide

HDL Usage Through Component Instantiation - VHDL

```
------------------------------------------------------------------------------
--
--  This example uses an instance of DW_lp_pipe_mgr along with an instance
--  of DW_pl_reg to pipeline an unsigned multiplication operation.
--
------------------------------------------------------------------------------
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use DWARE.DW_Foundation_comp.all;
use DWARE.DWpackages.all;

entity DW_lp_pipe_mgr_inst is
      generic (
        inst_a_width : POSITIVE := 8;
        inst_b_width : POSITIVE := 8;
        inst_id_width : POSITIVE := 8;
        inst_stages : POSITIVE := 4
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;

        -- upstream (input) status
        pipe_full_inst : out std_logic;
        pipe_ovf_inst : out std_logic;

        -- upstream (input) interface
        -- (request, ID & data)
        inst_launch : in std_logic;
        inst_launch_id : in std_logic_vector(inst_id_width-1 downto 0);
        inst_a : in std_logic_vector(inst_a_width-1 downto 0);
        inst_b : in std_logic_vector(inst_b_width-1 downto 0);

        -- downstream (output) interface
        -- (output indication, ID * data)
        arrive_inst : out std_logic;
        arrive_id_inst : out std_logic_vector(inst_id_width-1 downto 0);
        product_inst : out std_logic_vector(inst_a_width+inst_b_width-1 downto 0);

        -- downstream (output) FIFO flow control
        inst_accept_n : in std_logic;
        push_out_n_inst : out std_logic;

        -- pipe content status
        pipe_census_inst : out std_logic_vector(bit_width(inst_stages+1)-1 downto 0)
        );
```

```
    end DW_lp_pipe_mgr_inst;


architecture inst of DW_lp_pipe_mgr_inst is

-- Embedded script causes registers to be balanced
--
-- synopsys dc_script_begin
-- set_optimize_registers "true"
-- synopsys dc_script_end


-- the UNSIGNED product
signal prod_int : UNSIGNED(inst_a_width+inst_b_width-1 downto 0);

-- copy of product that has been cast to std_logic_vector type
signal a_times_b : std_logic_vector(inst_a_width+inst_b_width-1 downto 0);

-- DW_lp_pipe_mgr will conrtol register enables to pipe
-- via a bus of enables (one per register stage)
--
signal local_enables : std_logic_vector(inst_stages-1 downto 0);

signal tie_high : std_logic;

begin

    tie_high <= '1';


    -- operation to be pipelined is just "a * b"
    --
    prod_int <= UNSIGNED(inst_a) * UNSIGNED(inst_b);

    a_times_b <= STD_LOGIC_VECTOR(prod_int);

    -- Instance of DW_lp_pipe_mgr
    U1 : DW_lp_pipe_mgr
    generic map (
          stages => inst_stages,
          id_width => inst_id_width
          )
    port map (
          clk => inst_clk,
          rst_n => inst_rst_n,
          init_n => tie_high,
          launch => inst_launch,
          launch_id => inst_launch_id,
          pipe_full => pipe_full_inst,
```

```
            pipe_ovf => pipe_ovf_inst,
            pipe_en_bus => local_enables,
            accept_n => inst_accept_n,
            arrive => arrive_inst,
            arrive_id => arrive_id_inst,
            push_out_n => push_out_n_inst,
            pipe_census => pipe_census_inst
            );

      -- An instance of DW_pl_reg is used to get the data registers
      -- in the pipe (initially stacked all at the output - but DC
      -- will balance them)  Note that DW_pl_reg will ungroup itself
      -- automatically so that its registers will be free to be
      -- balanced into the datapath logic.
      --
      U2 : DW_pl_reg
      generic map (
            width => inst_a_width+inst_b_width,
            in_reg => 0,
            stages => inst_stages,
            out_reg => 1,
            rst_mode => 0
            )
      port map (
            clk => inst_clk,
            rst_n => inst_rst_n,
            enable => local_enables,
            data_in => a_times_b,
            data_out => product_inst
            );

  end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
////////////////////////////////////////////////////////////////////////////////
//
//  This example uses an instance of DW_lp_pipe_mgr along with an instance
//  of DW_pl_reg to pipeline an unsigned multiplication operation.
//
////////////////////////////////////////////////////////////////////////////////

module DW_lp_pipe_mgr_inst( inst_clk, inst_rst_n,

          pipe_full_inst, pipe_ovf_inst,// upstream (input) status

          inst_launch, inst_launch_id,// upstream (input) interface
          inst_a, inst_b,  // (request, ID & data)

          arrive_inst, arrive_id_inst,// downstream (outptu) interface
          product_inst,    // (output indication, ID & data)

          inst_accept_n, push_out_n_inst,// downstream (output) FIFO flow control

          pipe_census_inst // pipe content status

// Embedded script causes registers to be balanced
//
// synopsys dc_script_begin
// set_optimize_registers "true"
// synopsys dc_script_end

          );


parameter inst_a_width = 8;
parameter inst_b_width = 8;
parameter inst_id_width = 3;
parameter inst_stages = 4;


//  For this application, the census bus will need to be
//  wide enough to carry a value equal to 'inst_stages'
//  So, define it to be ceil(log2(inst_stages+1))
`define census_width 3

`define prod_width  (inst_a_width+inst_b_width)

input                       inst_clk;
input                       inst_rst_n;

output                      pipe_full_inst;
```

```
output                      pipe_ovf_inst;

input                       inst_launch;
input [inst_id_width-1 : 0] inst_launch_id;
input [inst_a_width-1 : 0] inst_a;
input [inst_b_width-1 : 0] inst_b;

output arrive_inst;
output [inst_id_width-1 : 0] arrive_id_inst;
output [`prod_width-1 : 0] product_inst;

input inst_accept_n;
output push_out_n_inst;

output [(`census_width)-1 : 0] pipe_census_inst;



wire [`prod_width-1 : 0] a_times_b;

// DW_lp_pipe_mgr will control register enables to pipe
//
wire [inst_stages-1 : 0] local_enables;


    // operation to be pipelined is just "a * b"
    //
    assign a_times_b = inst_a * inst_b;


    // Instance of DW_lp_pipe_mgr
    DW_lp_pipe_mgr #(inst_stages, inst_id_width) U1 (
                .clk(inst_clk),
                .rst_n(inst_rst_n),
                .init_n(1'b1),
                .launch(inst_launch),
                .launch_id(inst_launch_id),
                .pipe_full(pipe_full_inst),
                .pipe_ovf(pipe_ovf_inst),
                .pipe_en_bus(local_enables),
                .accept_n(inst_accept_n),
                .arrive(arrive_inst),
                .arrive_id(arrive_id_inst),
                .push_out_n(push_out_n_inst),
                .pipe_census(pipe_census_inst) );

    // An instance of DW_pl_reg is used to get the data registers
    // in the pipe (initially stacked all at the output - but DC
    // will balance them)  Note that DW_pl_reg will ungroup itself
```

```
      // automatically so that its registers will be free to be
      // balanced into the datapath logic.
      //
      DW_pl_reg #(`prod_width, 0, inst_stages, 1, 0) U2 (
                  .clk(inst_clk),
                  .rst_n(inst_rst_n),
                  .enable(local_enables),
                  .data_in(a_times_b),
                  .data_out(product_inst) );

  endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
March 2019	DWBB_201903.0	■ Updated licensing requirements in Table 1-3 on page 2 ■ Added "Enabling minPower" on page 9 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com